# Debugging
# Art, Science: Neither, Either or Both?

Perry Kivolowitz
University of Wisconsin – Madison
Computer Sciences Department

## Claim: You probably suck at debugging

- Basis of claim:
  Years of helping students debug
- It's not entirely your fault
  Few teach how to debug

  Some say it is an art and cannot be taught



## My view

- Certainly there is an art to debugging
- Mastery matures over years of experience
- But the art is based on science
  - Understanding the science can accelerate the growth of your artistry
  - Science **can** be taught



## The Science of Debugging

1. The Scientific Method
4. Kivolowitz's Corollary
5. Kivolowitz's Maxim 1
6. Maxwell Cohen's Law
7. Kivolowitz's Maxim 2
8. Kivolowitz's Maxim 3
9. Conan Doyle's Law



## Case study from a CS class*

- (Single threaded) code ran well on two different machines
- Crashed on a third
- Third machine got OS patches
- Bug went away
- Students asked: Was it an OS bug?

* All scenarios in this talk happened … more or less



## Case study from a CS class

- No, it was theirs
- In fact:
  - Symptoms tell you the type of bug it likely is
  - Symptoms explain why the bug went into hiding (Heisenbug)

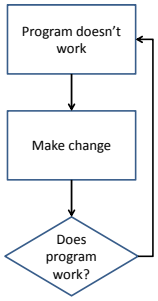
(Note to self: Give other examples of Heisenbugs)

## Case study from a CS class

- The bug is likely a pointer problem
  - Using an uninitialized pointer
  - Using a pointer after what it pointed to was freed
  - Using a pointer clobbered by something else
- Why did an OS patch "fix" the bug?
  - Because memory shifted around
  - The bogus pointer now points to a less "crash inducing" value (or the clobbered location…)
- The bug is still there, waiting…



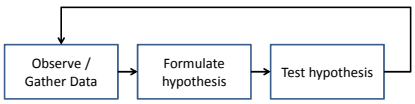
## Common Debugging Algorithm






## Principles 1 through 3

- Missing from the preceding is application of the Scientific Method




- Debugging should not be knee-jerk

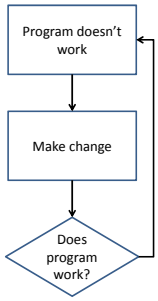


## Kivolowitz Corollary

# A fix is not a fix until you completely understand why it is a fix



## What else is wrong with this?






## Kivolowitz Maxim #1

If you make a change with no beneficial result – back it out unless you are certain you are fixing a different bug

Else, you are chasing a moving target

Debugging is about the elimination of unknowns, not their introduction.

## Maxwell (Mickey) Cohen's Law

"Where there is one, there are likely many"

- Always be mindful that you may be seeing a cascade of errors
  - Revisited during "defensive programming" and "call stack" discussion

(Note to self: Describe how debugging is different from looking for your phone)



## Kivolowitz Maxim #2

Student: It worked yesterday. Then I wrote X00 lines of code and now it doesn't work. Can you help me?

Master: Sigh.

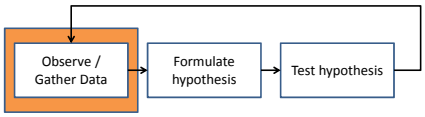
Write in small units.
Test in small units.

(Note to self: Even "personal" version control is useful)



## Kivolowitz's Maxim #3

- Bugs want to be found
- They typically announce themselves clearly if you make the effort to listen






## Kivolowitz's Maxim #3 Example

Student: I get "Bus error. Core dumped." Can you help me?

Master: What do you think the problem is?

Student: How should I know? That's all it says.

Master: Sigh.


The Guru Meditation from arguably the most bug-free non-trivial OS in history. This contains a great deal of information.



## Conan Doyle's Law

"When you have eliminated the impossible, whatever remains, however improbable, must be the truth."

The Sign of Four, 1890



## Defensive programming

- Cohen's law and Conan Doyle's law segue into defensive programming
- Both suggest the use of <assert.h>
- If you "know" a condition to be true, assert it!
- Stops a cascade in its tracks
- Helps to eliminate the impossible and identify the improbable

## Defensive programming

- Code abstraction can be defensive programming
- Code is more easily debugged when you…
  - Code in one place
  - Test in one place
  - Fix in one place
- Aside: leverage STL, Boost, glm etc.
  - Templates ("generics") in general
- Aside: code the "default" case and the impossible "else"

(Note to self: Emphasize "fix in one place" with respect to cut and pasted code)
(Note to self: "Improbable else" – testing floats and doubles – remember CS 252, CS 354 and CS 412)



## Defensive programming

- Comments
  - Don't just comment the code
  - Comment the thought process
    - As you write down your thoughts you get clarity
    - Will you remember what you were thinking a year from now? Next week? Tomorrow?

      What about someone else reading your code?

(Note to self: Mention value of writing the comments EVEN BEFORE writing the code!)



## Defensive programming

Student: I commented like you told me.

```
// Increment j
```

Master: Sigh.

Comment your thought process not minutia.



## Defensive programming

Student: My code doesn't work.
Can you help?

Master: What does variable `ineedsleep` do?

Student: I don't remember, I was tired.

Master: Sigh.

`descriptive_variable_names_really_help`

Keystrokes are cheap. Debugging time isn't.

(Note to self: Explain … keystrokes are guaranteed to end, debugging hours are not.)



## Role of testing

Testing can only prove the presence
of bugs, not their absence.

Edsger W. Dijkstra

Beware of bugs in the above code; I have
only proved it correct, not tried it.

Donald Knuth

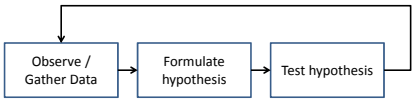
*Beware of bugs in the above code;*
*I have only written it, not tried it*



## Role of testing

- If you are fortunate to have a QA group
  - It is not their job to *find* bugs
  - That is your job
  - It is their job to confirm bugs you found are gone
- Don't "throw your code over a wall"

## Debugging tools and techniques




The scientific method is your strongest, most versatile tool



## Debugging tools and techniques

- Language based debugging aids
  - Use strong typing whenever possible
  - Pass structures and instances by reference via "&" instead of "*" where possible
  - (Use destructors to) munge pointers to recognizable values
    - Historical reference to DEADBEEF
  - "Managed" languages and runtimes are sweet

(Note to self: Mention `void *` is a fickle friend)

(Note to self: don't depend on side effects – remember register based param passing)



## Debugging tools and techniques

- Language based debugging aids
  - Use preprocessor for compile time / run time aid
    `#if`
    `#ifdef`
  - Examples:
    `#ifdef _DEBUG`
    And the verbose:
    `#if DEBUG_VERBOSITY > 5`
  - Non-destructive experimentation



## Debugging tools and techniques

- Single step
- Breakpoints
  - Always
  - Conditional
    - In debugger
    - In code
- Call stack
- Re-execution of code
- Immediate modes
- Value inspection
- Value modification

(Note to self: Spend a good deal of time on this slide – it is really important)



## Debugging tools and techniques

Student: My code doesn't work. Can you help me?

Master: Have you checked all return values?

Student: No.

Master: Sigh.

Return values are meant to be checked.



## Debugging tools and techniques

Student: My code doesn't work. Can you help me?

Master: Have you checked all return values?

Student: No. Checking makes my code messy.

Master: Sigh.

Exceptions (also) make handling errors "cleaner."

## Debugging tools and techniques

Student: My program stops responding to me. Can you help?
Master: Do you have an infinite loop?
Student: I don't know.
Master: Sigh.

Monitor CPU usage to help uncover infinite loops.
Asserts and console output can also help.

(Note to self: Oh how I miss my Altair and PDPs – Das Blinkenlights)
(Note to self: while (true) with no break, for / while i < x without changing i, unsigneds etc.)



## Debugging tools and techniques

Student: My program worked great. Then it crashed. Can you help me?
Master: Have you watched memory usage?
Student: No.
Master: Sigh.

Check memory consumption.
Leave code running for days if need be.
Use leak detection tools.

Use automated user "simulation" tools if appropriate. Fuzz testing (Prof. B Miller)



## Debugging tools and techniques

Student: This code doesn't do anything. Can you help me?
Master: Do you know if it is even executed?
Student: No.
Master: Sigh.

Console output, breakpoints and code coverage tools all help.

(Note to self: Debugging the wrong executable is not a productive use of time)



## Debugging tools and techniques

Student: My program crashes when I give it this input. Can you help me?
Master: Have you traced execution with that input?
Student: No.
Master: Sigh.

Vet and clean (all) input data to degree affordable.
Defensive programming!

(Note to self: Remind audience that idiots* [and bad guys] can be "smarter" than they are)
(*idiots as in the expression "Idiot proof")



## Debugging tools and techniques

Student: I must have a compiler / OS bug. Can you help me?
Master: Have you written a minimal test harness that manifests the bug?
Student: No.
Master: Sigh.

A minimal test harness saves time (and face).
See "Write in small units – Test in small units"

(Note to self: X00,000,000 other users / coders and YOU are the one to find it? Not THAT often!)



## Debugging tools and techniques

Student: My code crashes every time it gets here!
Master: May you always be so fortunate.
Student: [under breath] What a jerk!

Repeatable bugs are your friends.
Can you devise a test that makes a bug repeatable? ("Break it to fix it")

## Personal comments

A technique I use frequently:

1. Punt. Ask for help via email
2. Feel ashamed – did I punt too soon?
3. Try again (using techniques described here)
4. Fix bug … mumble "*I am such an idiot*"
5. Send email saying "Never mind…"



## Seriously…

- You will find yourself saying "I am such an idiot" many many times over your career
- Internalize:
  - Yes, in that context, you were an idiot
  - We ALL are, every one of us.
    I am an idiot typically several times a day.
  - IT's OK! Bugs happen!
  - If coding is your passion, it may help to think of debugging as part of perfecting your creation!

Could be a riddle: What do you try really hard to avoid then really hard to find?



## Last, but not least

Student: I have an error. Can you help me?
Master: What did the error message say?
Student: I don't know. I didn't read it.
Master: We're done here.

Thank you!